FIG. 1

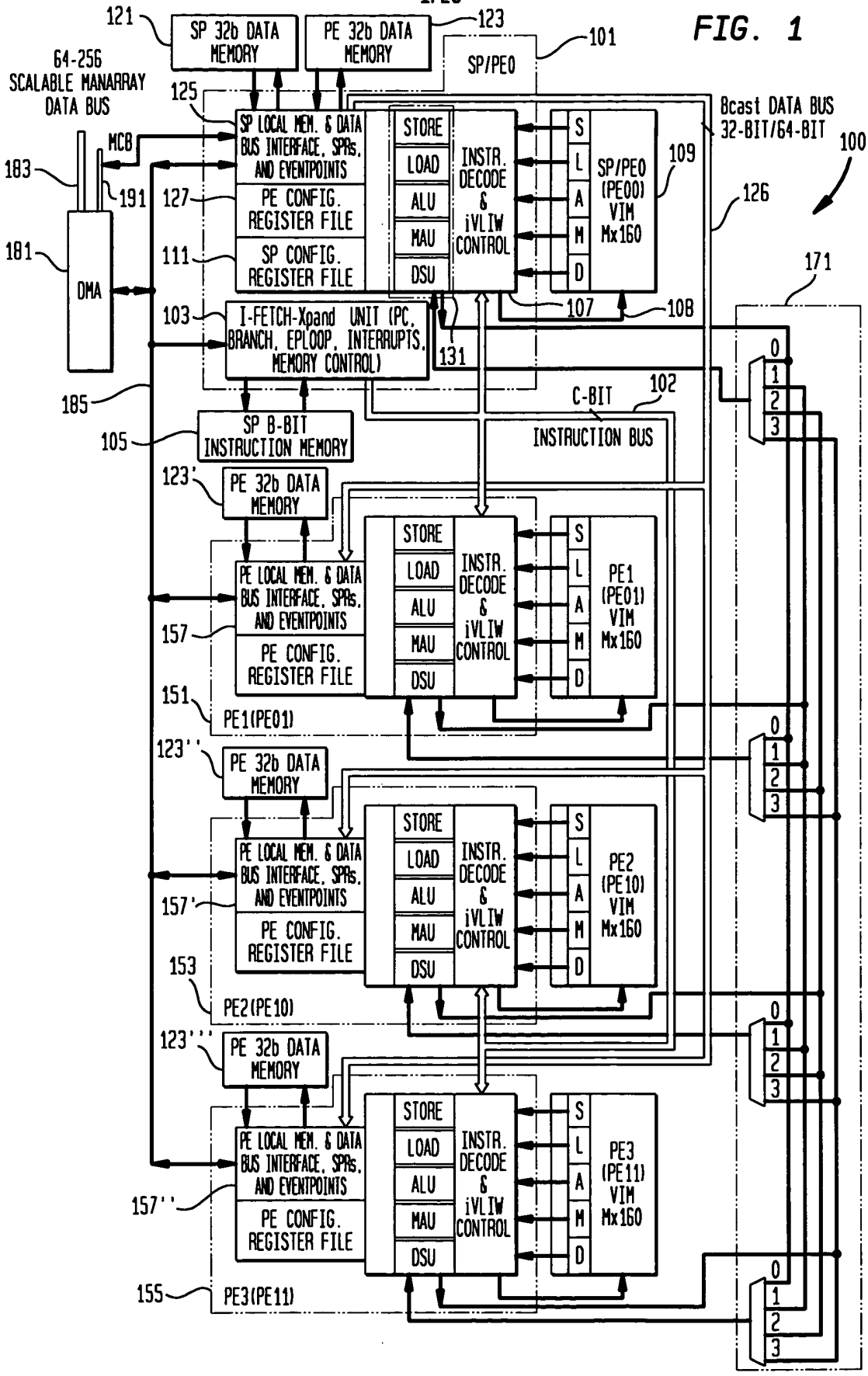
121
SP 32b DATA MEMORY
123
PE 32b DATA MEMORY
101
SP/PE0
64-256 SCALABLE MANARRAY DATA BUS
125
SP LOCAL MEM. & DATA BUS INTERFACE, SPRs, AND EVENTPOINTS
STORE
LOAD
ALU
MAU
DSU
INSTR. DECODE & iVLIW CONTROL
S
L
A
M
D
SP/PE0 (PE00) VIM Mx160
109
8cast DATA BUS 32-BIT/64-BIT
100
MCB
183
191
127
PE CONFIG. REGISTER FILE
126
181
DMA
111
SP CONFIG. REGISTER FILE
171
103
I-FETCH-Xpand UNIT (PC, BRANCH, EPLOOP, INTERRUPTS, MEMORY CONTROL)
107
108
131
185
C-BIT
102
105
SP B-BIT INSTRUCTION MEMORY
INSTRUCTION BUS
123'
PE 32b DATA MEMORY
PE LOCAL MEM. & DATA BUS INTERFACE, SPRs, AND EVENTPOINTS
157
PE CONFIG. REGISTER FILE
PE1 (PE01) VIM Mx160
151
PE1(PE01)
123''
PE 32b DATA MEMORY
157'
PE2 (PE10) VIM Mx160
153
PE2(PE10)
123'''
PE 32b DATA MEMORY
157''
PE3 (PE11) VIM Mx160
155
PE3(PE11)
0
1
2
3

FIG. 2A

200

LSPR ENCODING

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUP		S/P	L/S	110			CE1	SIZE		0	Rt					0	0	0	0	0	0	SPRADDR									

FIG. 2B

210

Syntax/Operation

Instruction	Operands	Operation
LOAD ADDRESS REGISTER		
LSPR.[SP].W	Rt, SPRADDR	Rt ← [SPRADDR]word
LSPR.[SP].H0	Rt, SPRADDR	Rt.H0 ← [SPRADDR]hword
LSPR.[SP].B0	Rt, SPRADDR	Rt.B0 ← [SPRADDR]byte
T.LSPR.[SP].[WH0B0]	Rt, SPRADDR	Do operation only if T condition is satisfied in F0

FIG. 3A

300

SSPR ENCODING

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUP		S/P	L/S	110			CE1	SIZE		0	Rt					0	0	0	0	0	0	SPRADDR									

FIG. 3B

310

Syntax/Operation

Instruction	Operands	Operation
STORE SPECIAL-PURPOSE REGISTER		
SSPR.[SP].W	Rt, SPRADDR	Rt → [SPRADDR]word
SSPR.[SP].H0	Rt, SPRADDR	Rt.H0 → [SPRADDR]hword
SSPR.[SP].B0	Rt, SPRADDR	Rt.B0 → [SPRADDR]byte
T.SSPR.[SP].[WH0B0]	Rt, SPRADDR	Do operation only if T condition is satisfied in F0

401 403 405 407 SPR REGISTER MAP 409 FIG. 4

400

SYSTEM ADDRESS FOR SP AS A MCB BUS MASTER (THE NON-SP MCB ADDRESS IS THE SAME EXCEPT WITH ADDRESS BIT 22=1)	SP/PE LSPR/ SSPR ADDRESS	SP REGISTER	PE REGISTER	DESCRIPTION
0x0030080	0x0080	DBSTAT		Debug Status register
0x0030084	0x0084	DBIR		Debug Instruction register
0x0030088	0x0088	DBDIN		Debug Data In register
0x003008c	0x008c	DBDOUT		Debug Data Out register
0x0030090	0x0090			
0x0030094	0x0094			
0x0030098	0x0098			
0x003009c	0x009c			
0x00300a0	0x00a0	EPSTAT	EPSTAT	Event Point Status
0x00300a4	0x00a4	DEPCTL0	DEPCTL0	Data Event point Control register 0
0x00300a8	0x00a8	DEPCTL1		Data Event point Control register 1
0x00300ac	0x00ac	DEP0R0	DEP0R0	Data Event point 0 register 0
0x00300b0	0x00b0	DEP0R1	DEP0R1	Data Event point 0 register 1
0x00300b4	0x00b4	DEP0R2	DEP0R2	Data Event point 0 register 2
0x00300b8	0x00b8	DEP1R0		Data Event point 1 register 0
0x00300bc	0x00bc	DEP1R1		Data Event point 1 register 1
0x00300c0	0x00c0	DEP1R2		Data Event point 1 register 2
0x00300c4	0x00c4	DEP2R0		Data Event point 2 register 0
0x00300c8	0x00c8	DEP2R1		Data Event point 2 register 1
0x00300cc	0x00cc	DEP2R2		Data Event point 2 register 2
0x00300d0	0x00d0	IEPCTL0		Instruction Event point Control register 0
0x00300d4	0x00d4	IEPCTL1		Instruction Event point Control register 1
0x00300d8	0x00d8	IEP0R0		Instruction Event point 0 register 0
0x00300dc	0x00dc	IEP0R1		Instruction Event point 0 register 1
0x00300e0	0x00e0	IEP0R2		Instruction Event point 0 register 2
0x00300e4	0x00e4	IEP1R0		Instruction Event point 1 register 0
0x00300e8	0x00e8	IEP1R1		Instruction Event point 1 register 1
0x00300ec	0x00ec	IEP1R2		Instruction Event point 1 register 2
0x00300f0	0x00f0	IEP2R0		Instruction Event point 2 register 0
0x00300f4	0x00f4	IEP2R1		Instruction Event point 2 register 1
0x00300f8	0x00f8	IEP2R2		Instruction Event point 2 register 2
0x00300fc	0x00fc	IEP3R0		Instruction Event point 3 register 0
0x0030100	0x0100	IEP3R1		Instruction Event point 3 register 1
0x0030100	0x0104	IEP3R2		Instruction Event point 4 register 2
0x0030100	0x0108	IEP4R0		Instruction Event point 4 register 0
0x0030100	0x010c	IEP4R1		Instruction Event point 4 register 1
0x0030100	0x0110	IEP4R2		Instruction Event point 4 register 2
0x0030100	0x0114	IEP5R0		Instruction Event point 5 register 0
0x0030100	0x0118	IEP5R1		Instruction Event point 5 register 1
0x0030100	0x011c	IEP5R2		Instruction Event point 5 register 2

410

FIG. 5

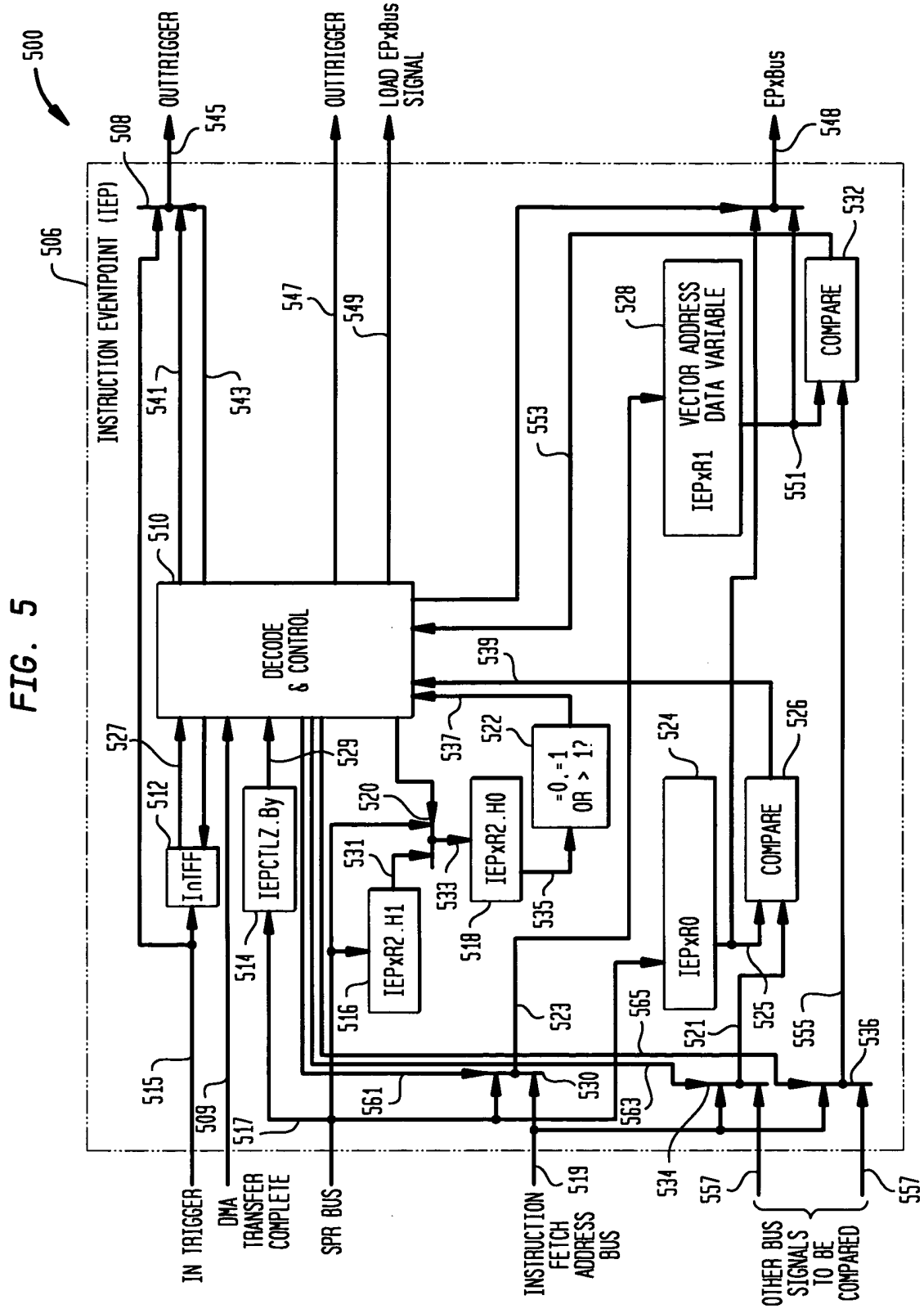
500
INSTRUCTION EVENTPOINT (IEP)
506
OUTTRIGGER
508
545
OUTTRIGGER
547
LOAD EPxBus SIGNAL
549
EPxBus
548
541
543
553
528
VECTOR ADDRESS DATA VARIABLE
IEPxR1
551
532
COMPARE
510
DECODE & CONTROL
539
527
512
InTFF
529
IEPCTLZ.By
514
520
531
533
IEPxR2.H1
516
518
IEPxR2.H0
522
537
=0, =1 OR > 1?
535
524
IEPxR0
526
COMPARE
523
565
521
525
555
536
515
509
517
561
530
563
534
557
557
IN TRIGGER
DMA TRANSFER COMPLETE
SPR BUS
INSTRUCTION FETCH ADDRESS BUS
519
OTHER BUS SIGNALS TO BE COMPARED

FIG. 6A

602 CONTROL VALUE	601 OPERATION (600)
00000000	Disabled Event point. No action OutTrigger ← InTrigger; — 603

604 — 00T11000
 	
This may be used for a loop, with loop skip if count is zero.
If T==1 InTrigger can be used to exit or skip the loop.
	OutTrigger ← InTrigger; //always pass trigger through — 605
	
	if(T==1) — 606
	InTriggerFF ← InTrigger; — 607
	else — 608
	InTriggerFF ← 1; — 609
	
	WHEN((PC==IEPxR0||PC==IEPxR1)) — 610
	{
	if((PC==IEPxR1)&& //if PC matches "start" address... — 611
	((T==1&&InTriggerFF==1)|| //trigger is enabled and active — 612
	(IEPxR2.H0==0) //OR 'count' is zero — 613
	)
	)
	{
	PC ← IEPxR0; //jump to end of loop — 614
	IEPxR2.H0 ← IEPxR2.H1; //reload count — 615
	InTriggerFF ← 0; //clear FF — 616
	CancelNext(Inst[PC]); //Cancel last next fetched inst — 617
	//(last inst of loop)
	}
	else if(PC==IEPxR0) //if at "end" address — 618
	{
	if(T==1 && InTriggerFF==1) //if trigger enabled and active — 619
	{
	PC ← next PC; //fall out of loop — 620
	IEPxR2.H0 ← IEPxR2.H1; //reload count — 621
	InTriggerFF ← 0; //clear FF — 622
	}
	else if(IEPxR2.H0==0||IEPxR2.H0==1) — 623
	{
	PC ← next PC; — 624
	IEPxR2.H0 ← IEPxR2.H1; — 625
	}
	else //else Count is neither 1 nor 0 — 626
	{
	PC ← IEPxR1; //next PC is [IEPxR1] — 627
	IEPxR2.H0 ← IEPxR2.H0-1; //decrement "count" — 628
	}
	}
	
	)

FIG. 6B

602 601 640

CONTROL VALUE	OPERATION

00T11001
 	
This is used for a loop which will not skip to the end if the start address is reached and the count is zero.
If T==1 Trigger can be used to exit or skip the loop.
	OutTrigger ← InTrigger; //always pass trigger through
	
	if(T==1)
	InTriggerFF ← InTrigger;
	else
	InTriggerFF ← 1;
	
	WHEN((PC==IEPxR0||PC==IEPxR1))
	{
	if((PC==IEPxR1)&& //if PC matches "start" address...
	(T==1&&InTriggerFF==1) //trigger is enabled and active
	)
	{
	PC ← IEPxR0; //jump to end of loop
	IEPxR2.H0 ← IEPxR2.H1; //reload count
	InTriggerFF ← 0; //clear FF
	CancelNext(Inst[PC]); //Cancel last next fetched inst
	//(last inst of loop)
	}
	else if(PC==IEPxR0) //if at "end" address
	{
	if(T==1 && InTriggerFF==1) //if trigger enabled and active
	{
	PC ← next PC; //fall out of loop
	IEPxR2.H0 ← IEPxR2.H1; //reload count
	InTriggerFF ← 0; //clear FF
	}
	else if(IEPxR2.H0==0||IEPxR2.H0==1)
	{
	PC ← next PC;
	IEPxR2.H0 ← IEPxR2.H1;
	}
	else //else Count is neither 1 nor 0
	{
	PC ← IEPxR1; //next PC is [IEPxR1]
	IEPxR2.H0 ← IEPxR2.H0-1; //decrement "count"
	}
	}
	}

FIG. 6C

602 601 650

CONTROL VALUE	OPERATION

00T11010
 	
This may be used for a loop with an exit based on a TRUE F0 condition or count == 0, or pre-trigger (if enabled).
If enabled for pre-trigger, and trigger FF is set, and 'start' address is matched, then the loop is skipped.
	OutTrigger ← InTrigger; //always pass trigger through
	if(T==1)
	InTriggerFF ← InTrigger;
	else
	InTriggerFF ← 1;
	
	WHEN((PC==IEPxR0||PC==IEPxR1))
	{
	if((PC==IEPxR1)&& //if PC matches "start" address...
	(T==1&&InTriggerFF==1) //trigger is enabled and active
	)
	{
	PC ← IEPxR0; //jump to end of loop
	IEPxR2.H0 ← IEPxR2.H1; //reload count
	InTriggerFF ← 0; //clear FF
	CancelNext(Inst[PC]); //Cancel last next fetched inst
	//(last inst of loop)
	}
	else if(PC==IEPxR0) //if at "end" address
	{
	if(T==1 && InTriggerFF==1) //if trigger enabled and active
	{
	PC ← next PC; //fall out of loop
	IEPxR2.H0 ← IEPxR2.H1; //reload count
	InTriggerFF ← 0; //clear FF
	}
	else if(IEPxR2.H0==0||IEPxR2.H0==1||F0==1
	{
	PC ← next PC;
	if(IEPxR2.H0==0||IEPxR2.H0==1)
	{
	IEPxR2.H0 ← IEPxR2.H1;
	}
	}
	else //else Count is neither 1 nor 0
	{
	PC ← IEPxR1; //next PC is [IEPxR1]
	IEPxR2.H0 ← IEPxR2.H0-1; //decrement "count"
	}
	}
	}

FIG. 6D

602 CONTROL VALUE	601 OPERATION 660

00T11011
 	
This may be used for a loop with an exit based on a FALSE F0 condition or count == 0, or pre-trigger (if enabled).
If enabled for pre-trigger, and trigger FF is set, and 'start' address is matched, then the loop is skipped.
	OutTrigger ← InTrigger; //always pass trigger through
	
	if(T==1)
	InTriggerFF ← InTrigger;
	else
	InTriggerFF ← 1;
	
	WHEN((PC==IEPxR0||PC==IEPxR1))
	{
	if((PC==IEPxR1) //if PC matches "start" address...
	(T==1&&InTriggerFF==1) //trigger is enabled and active
	)
	{
	PC ← IEPxR0; //jump to end of loop
	IEPxR2.H0 ← IEPxR2.H1; //reload count
	InTriggerFF ← 0; //clear FF
	CancelNext(Inst[PC]); //Cancel last next fetched inst
	//(last inst of loop)
	}
	}
	else if(PC==IEPxR0) //if at "end" address
	{
	if(T==1 && InTriggerFF==1) //if trigger enabled and active
	{
	PC ← next PC; //fall out of loop
	IEPxR2.H0 ← IEPxR2.H1; //reload count
	InTriggerFF ← 0; //clear FF
	}
	else if(IEPxR2.H0==0||IEPxR2.H0==1||F0==0)
	{
	PC ← next PC;
	if(IEPxR2.H0==0||IEPxR2.H0==1)
	{
	IEPxR2.H0 ← IEPxR2.H1;
	}
	}
	else //else Count is neither 1 nor 0
	{
	PC ← IEPxR1; //next PC is [IEPxR1]
	IEPxR2.H0 ← IEPxR2.H0-1; //decrement "count"
	}
	}
	}

FIG. 6E

602 601 670

CONTROL VALUE	OPERATION

SPT00000
 	
Used for generating an EP interrupt with optional pre-count, and pre-trigger. Interrupt occurs just after instruction at address IEPxR1.
 	
If SPT=000, then this option becomes the "No operation" control code and no updates to IEP registers occur.
	if(P==0) //Default is pass-through
	OutTrigger ← InTrigger;
	else //Default is "no trigger"
	OutTrigger ← 0;
	if(T==1)
	InTriggerFF ← InTrigger;
	else
	InTriggerFF ← 1;
	
	WHEN((PC==IEPxR0)||(PC==IEPxR1)&&(!T||InTriggerFF))
	{
	if(T && InTriggerFF)
	InTriggerFF ← 0;
	
	if(IEPxR2.H0 == 0) //If match with count=0, carry on
	{ //(acts as if disabled)
	PC ← next PC;
	}
	else if(IEPxR2.H0 == 1)
	{
	if(S==1)
	EPINT ← 1; //assert EP interrupt
	if(P==1)
	OutTrigger ← 1;
	IEPxR2.H0 ← IEPxR2.H1;
	}
	else
	{
	PC ← next PC;
	IEPxR2.H0 ← IEPxR2.H0 - 1;
	}
	}

FIG. 6F

602 601 680

CONTROL VALUE	OPERATION

OPT00001
 	
Used for vectoring to a target address after 'count' matches
	if(P==0)
	OutTrigger ← InTrigger; //Default is pass-through
	else
	OutTrigger ← 0; //Default is "no trigger"
	if(T==1)
	InTriggerFF ← InTrigger;
	else
	InTriggerFF ← 1;
	WHEN((PC==IEPxR0)&&(!T||InTriggerFF))
	{
	if(T && InTriggerFF) //If pre-trigger, then clear FF
	InTriggerFF ← 0;
	
	if(IEPxR2.H0 == 0) //If match with count=0, carry on
	{ //(acts as if disabled)
	PC ← next PC;
	}
	else if(IEPxR2.H0 == 1) //if Match with count 1
	{
	PC ← IEPxR1; //Branch to vector address
	IEPxR2.H0 ← IEPxR2.H1; //Reload 'count'
	}
	else //if neither 0 or 1...
	{
	PC ← next PC; //next PC
	IEPxR2.H0 ← IEPxR2.H0 - 1; //decrement 'count'
	}
	}

FIG. 6G

602 601 690

CONTROL VALUE	OPERATION

SPT00010
 	
This IEP can be used to generate an EP interrupt after 'count' input trigger events have been received. (If T==0, then the branch occurs after 'count' cycles, repeatedly)
	if(P==0)
	OutTrigger ← InTrigger; //Default is pass-through
	else
	OutTrigger ← 0; //Default is "no trigger"
	if(T==1)
	InTriggerFF ← InTrigger;
	else
	InTriggerFF ← 1;
	WHEN(InTriggerFF))
	{
	if(T && InTriggerFF) //If pre-trigger, then clear FF
	InTriggerFF ← 0;
	
	if(IEPxR2.H0 == 0) //If match with count=0, carry on
	{ //(acts as if disabled)
	PC ← next PC;
	}
	else if(IEPxR2.H0 == 1) //if Match with count 1
	{
	if(S==1)
	EPINT ← 1; //assert EP interrupt
	if(P==1)
	OutTrigger ← 1;
	IEPxR2.H0 ← IEPxR2.H1;
	}
	else //If neither 0 or 1...
	{
	PC ← next PC; //next PC
	IEPxR2.H0 ← IEPxR2.H0 - 1; //decrement 'count'
	}
	}

FIG. 7A

EPLOOPx
ENCODING

700

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUP		S/P	CtrlOp				LCF	BPID		0	0	0	0	0	0	0	0	0	0	0	0	UDISP10									

FIG. 7B

710

Syntax/Operation

Instruction	Operands	Operation

EPLOOPx	UDISP10	IEPxR1 ← PC + 1
		IEPxR0 ← PC + UDISP10
		IEPx ← 0x18
		if(IEPxR2.H0>0){
		  while(IEPxR2.H0>1){
		  Execute instructions until PC = IEPxR0
		  PC ← IEPxR1
		  IEPxR2.H0 ← IEPxR2.H0-1
		  }
		  Execute instructions until PC = IEPxR0
		  IEPxR2.H0 ← IEPxR2.H1
		}
		else
		  PC ← PC + UDISP10 + 1

FIG. 7C

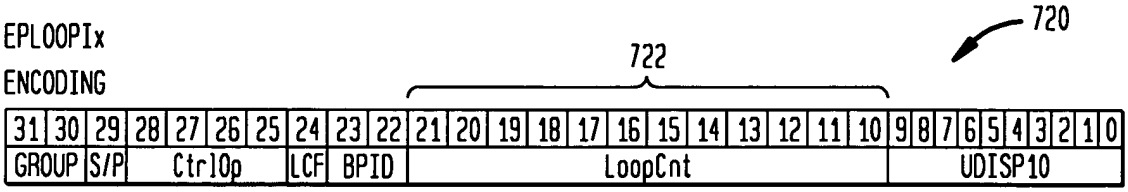

FIG. 7D

Syntax/Operation

730

Instruction	Operands	Operation

EPLOOPIx	LoopCnt, UDISP10,	IEPxR1 ← PC + 1
		IEPxR0 ← PC + UDISP10
		IEPxR2.H0 ← LoopCnt
		IEPxR2.H1 ← loopCnt
		IEPx ← 0x18
		if(IEPxR2.H0>0){
		  while(IEPxR2.H0>1){
		  Execute instructions until PC = IEPxR0
		  PC ← IEPxR1
		  IEPxR2.H0 ← IEPxR2.H0 - 1
		  }
		  Execute instructions until PC = IEPxR0
		  IEPxR2.H0 ← IEPxR2.H1
		}
		else
		  PC ← PC + UDISP10 + 1

FIG. 8

800

	802 Cycle	804 EP Compare	806 Fetch	808 Decode	810 Execute
812	0		EPLOOPx	Instruction before EPLOOPx	Second instruction before EPLOOPx
814	1		First instruction of Loop	EPLOOPx 1. Calculate EndA=PC+DISP 2. Hold PC and NOP instruction in fetch	Instruction before EPLOOPx
816	2		First instruction of Loop	HW NOP	EPLOOPx 1. Send EndA to IEPx on SPR bus to be loaded into IEPxR0 2. Signal IEPx to load the program counter value into IEPxR1 3. Hold PC and NOP instruciton in fetch
818	3	First compare of IEPxR0 to PC here.	First instruction of Loop	NOP	HW NOP
820	4		Next instruction	First instruction of Loop	HW NOP
822	5		⋮	Next instruction	First instruction of Loop

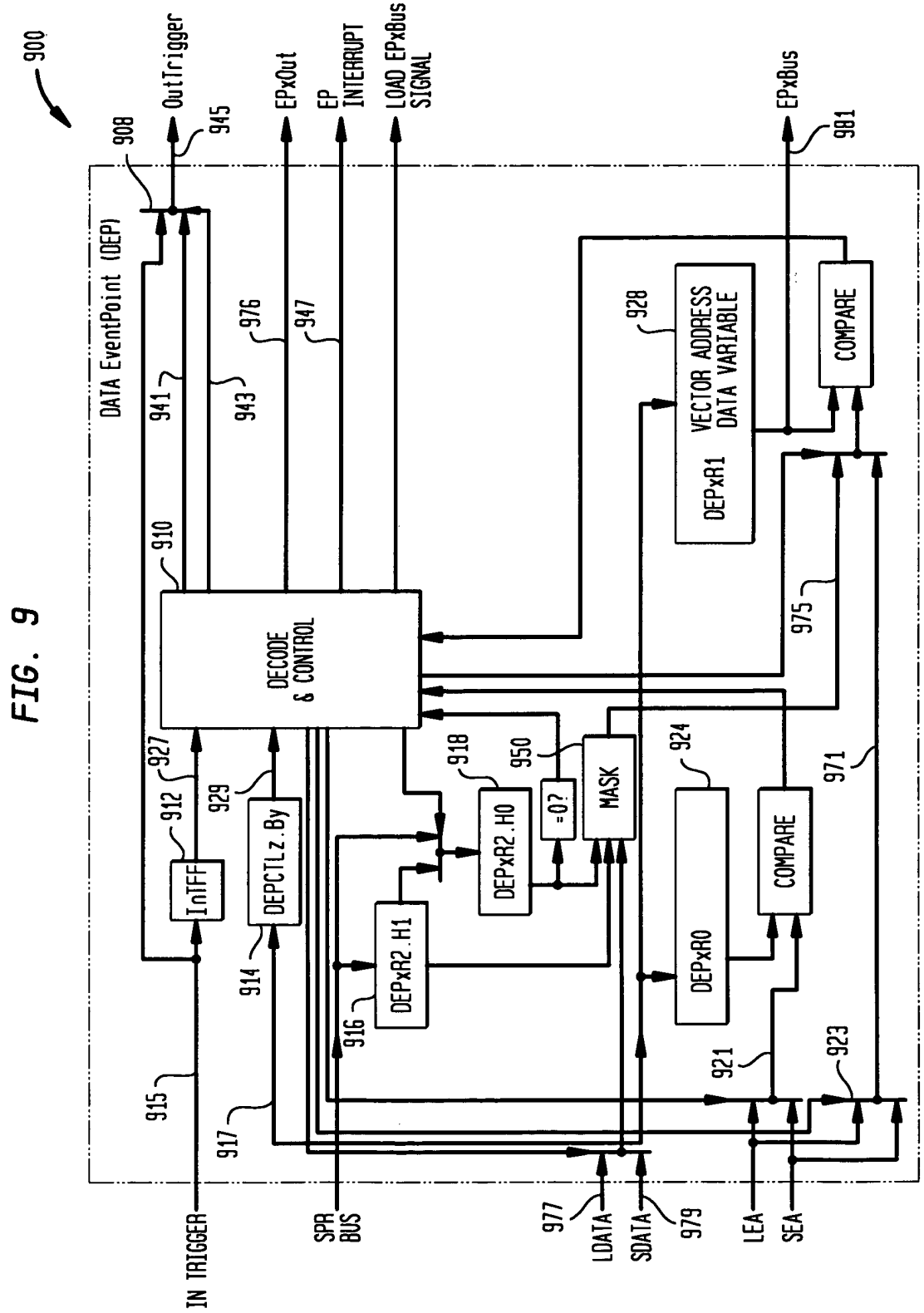
900
OutTrigger
945
EPxOut
EP INTERRUPT
LOAD EPxBus SIGNAL
EPxBus
981
908
DATA EventPoint (DEP)
941
943
976
947
928
VECTOR ADDRESS DATA VARIABLE
DEPxR1
COMPARE
910
DECODE & CONTROL
975
927
912
929
InTFF
DEPCTLz.By
914
916
DEPxR2.H1
DEPxR2.H0
918
950
=0?
MASK
924
DEPxR0
COMPARE
971
915
917
921
923
IN TRIGGER
SPR BUS
977
LDATA
SDATA
979
LEA
SEA

FIG. 9

FIG. 10A

1012 1010

CONTROL VALUE	OPERATION

SPT01000
 	
LEA match
&& (LDATA
& MASK)
match after 1
occurance then
interrupt or
OutTrigger
	if(P==0)
	OutTrigger ← InTrigger; //Default is pass-through
	else
	OutTrigger ← 0; //Default is "no trigger"
	if(T==1)
	InTriggerFF ← InTrigger;
	else
	InTriggerFF ← 1;
	
	WHEN((LEA==DEPxR0)&&((LDATA & DEPxR2)==DEPxR1)
	&& InTriggerFF))
	{
	if(T && InTriggerFF)
	InTriggerFF ← 0; //if qualified event, clear FF
	
	if(P==1)
	OutTrigger ← 1; //output 1 cycle pulse
	if(S==1)
	EPINT ← 1;
	
	}

SPT01001
 	
SEA match
&& (SDATA
& MASK)
match after 1
occurance then
interrupt or
OutTrigger
	if(P==0)
	OutTrigger ← InTrigger; //Default is pass-through
	else
	OutTrigger ← 0; //Default is "no trigger"
	if(T==1)
	InTriggerFF ← InTrigger;
	else
	InTriggerFF ← 1;
	
	WHEN((SEA==DEPxR0)&&((SDATA & DEPxR2)==DEPxR1)
	&& InTriggerFF))
	{
	if(T && InTriggerFF)
	InTriggerFF ← 0; //if qualified event, clear FF
	
	if(P==1)
	OutTrigger ← 1; //output 1 cycle pulse
	if(S==1)
	EPINT ← 1;
	
	}

1015

FIG. 10B

1012 ↘ 1020 ↙

CONTROL VALUE	OPERATION

SPT01010
 	
(LEA or SEA)
match after
count
occurances
then interrupt
or OutTrigger
	if(P==0)
	OutTrigger ← InTrigger; //Default is pass-through
	else
	OutTrigger ← 0; //Default is "no trigger"
	if(T==1)
	InTriggerFF ← InTrigger;
	Else
	InTriggerFF ← 1;
	WHEN((LEA==DEPxR0||SEA == DEPxR1)&&(InTriggerFF))
	{
	if(T && InTriggerFF) //If pre-trigger, then clear FF
	InTriggerFF ← 0;
	
	if(DEPxR2.H0==0) //If match with count=0, carry on
	{ //(acts as if disabled)
	PC ← next PC;
	}
	else if(DEPxR2.H0==1) //if Match with count 1...
	{
	if(P==1)
	OutTrigger ← 1; //output 1 cycle pulse
	if(S==1)
	EPINT ← 1;
	DEPxR2.H0 ← DEPxR2.H1; //Reload 'count'
	}
	else //If neither 0 or 1...
	{
	PC ← next PC;
	DEPxR2.H0 ← DEPxR2.H0 - 1; //decrement 'count'
	}
	
	}

FIG. 10C

1012 ↘ ↙ 1030

CONTROL VALUE	OPERATION

SPT01100
 	
(LEA THEN SEA) match after count occurances then interrupt or OutTrigger
	if(P==0)
	OutTrigger ← InTrigger; //Default is pass-through
	else
	OutTrigger ← 0; //Default is "no trigger"
	if(I==1)
	InTriggerFF ← InTrigger;
	else
	InTriggerFF ← 1;
	WHEN(((LEA==DEPxR0)&&(InTriggerFF))AND THEN(SEA == DEPxR1))
	{
	if(I && InTriggerFF) //If pre-trigger, then clear FF
	InTriggerFF ← 0;
	
	if(DEPxR2.H0==0) //If match with count 0, carry on
	{ //(acts as if disabled)
	PC ← next PC;
	}
	else if(DEPxR2.H0==1) //if Match with count 1...
	{
	if(P==1)
	OutTrigger ← 1; //output 1 cycle pulse
	if(S==1)
	EPINT ← 1;
	DEPxR2.H0 ← DEPxR2.H1; //Reload 'count'
	}
	else //If neither 0 or 1...
	{
	PC ← next PC;
	DEPxR2.H0 ← DEPxR2.H0 - 1 //decrement 'count'
	}
	}

FIG. 10D

1012 ↘ 1040 ↙

CONTROL VALUE	OPERATION

SPT01101
 	
(SEA THEN
LEA) match
after count
occurances
then interrupt
or OutTrigger
	if(P==0)
	OutTrigger ← InTrigger; //Default is pass-through
	else
	OutTrigger ← 0; //Default is "no trigger"
	if(T==1)
	InTriggerFF ← InTrigger;
	else
	InTriggerFF ← 1;
	WHEN(((SEA==DEPxR0)&&(InTriggerFF))AND THEN(LEA == DEPxR1))
	{
	if(T && InTriggerFF) //If pre-trigger, then clear FF
	InTriggerFF ← 0;
	
	if(DEPxR2.H0==0) //If match with count=0, carry on
	{ //(acts as if disabled)
	PC ← next PC;
	}
	else if(DEPxR2.H0==1) //if Match with count 1...
	{
	if(P==1)
	OutTrigger ← 1; //output 1 cycle pulse
	if(S==1)
	EPINT ← 1;
	DEPxR2.H0 ← DEPxR2.H1; //Reload 'count'
	}
	else //If neither 0 or 1...
	{
	PC ← next PC;
	DEPxR2.H0 ← DEPxR2.H0 - 1 //decrement 'count'
	}
	}

FIG. 10E

1012 ↘ 1050 ↙

CONTROL VALUE	OPERATION

SPT01110
 	
(LEA THEN LEA) match after count occurances then interrupt or OutTrigger
	if(P==0)
	OutTrigger ← InTrigger; //Default is pass-through
	else
	OutTrigger ← 0; //Default is "no trigger"
	if(T==1)
	InTriggerFF ← InTrigger;
	else
	InTriggerFF ← 1;
	WHEN(((LEA==DEPxR0)&&(InTriggerFF))AND THEN(LEA == DEPxR1))
	{
	if(T && InTriggerFF) //If pre-trigger, then clear FF
	InTriggerFF ← 0;
	
	if(DEPxR2.H0==0) //If match with count 0, carry on
	{ //(acts as if disabled)
	PC ← next PC;
	}
	else if(DEPxR2.H0==1) //if Match with count 1...
	{
	if(P==1)
	OutTrigger ← 1; //output 1 cycle pulse
	if(S==1)
	EPINT ← 1;
	DEPxR2.H0 DEPxR2.H1; //Reload 'count'
	}
	else //If neither 0 or 1...
	{
	PC ← next PC;
	DEPxR2.H0 ← DEPxR2.H0 - 1 //decrement 'count'
	}
	}

FIG. 10F

1012 → CONTROL VALUE

1060 → OPERATION

CONTROL VALUE	OPERATION

SPT01111
 	
(SEA THEN
SEA) match
after count
occurances
then interrupt
or OutTrigger
	if(P==0)
	OutTrigger ← InTrigger; //Default is pass-through
	else
	OutTrigger ← 0; //Default is "no trigger"
	if(T==1)
	InTriggerFF ← InTrigger;
	else
	InTriggerFF ← 1;
	WHEN(((SEA==DEPxR0)&&(InTriggerFF))AND THEN(SEA == DEPxR1))
	{
	if(T && InTriggerFF) //If pre-trigger, then clear FF
	InTriggerFF ← 0;
	
	if(DEPxR2.H0==0) //If match with count 0, carry on
	{ //(acts as if disabled)
	PC ← next PC;
	}
	else if(DEPxR2.H0==1) //if Match with count 1...
	{
	if(P==1)
	OutTrigger ← 1; //output 1 cycle pulse
	if(S==1)
	EPINT ← 1;
	DEPxR2.H0 ← DEPxR2.H1; //Reload 'count'
	}
	else //If neither 0 or 1...
	{
	PC ← next PC;
	DEPxR2.H0 ← DEPxR2.H0 - 1 //decrement 'count'
	}
	}

FIG. 10G

1012 ↘ 1070 ↙

CONTROL VALUE	OPERATION

SPT10000
 	
LDATA & MASK match after 1 occurances capture address then interrupt and/or OutTrigger. Once the address is held in DEPxR0, it remains held until the control value changes or a store to DEPxR0 occurs.
	if(P==0)
	OutTrigger ← InTrigger; //Default is pass-through
	else
	OutTrigger ← 0; //Default is "no trigger"
	if(T==1)
	InTriggerFF ← InTrigger;
	else
	InTriggerFF ← 1;
	
	DEPxR0 ← LEA; //save address where data detected
	
	WHEN((LDATA & DEPxR2) == DEPxR1
	&& InTriggerFF)
	{
	if(T && InTriggerFF)
	InTriggerFF ← 0; //if qualified event, clear FF
	
	Hold(DEPxR0); //Retain state of DEPxR0 (LEA)
	
	if(P==1)
	OutTrigger ← 1; //output 1 cycle pulse
	if(S==1)
	EPINT ← 1;
	
	}

FIG. 10H

1012 ↘ 1080 ↙

CONTROL VALUE	OPERATION

SPT10001
 	
SDATA & MASK match after 1 occurances capture address then interrupt and/or OutTrigger. Once the address is held in DEPxR0, it remains held until the control value changes or a store to DEPxR0 occurs.
	if(P==0)
	OutTrigger ← InTrigger; //Default is pass-through
	else
	OutTrigger ← 0; //Default is "no trigger"
	if(T==1)
	InTriggerFF ← InTrigger;
	else
	InTriggerFF ← 1;
	
	DEPxR0 ← SEA; //save address where data detected
	
	WHEN((SDATA & DEPxR2) == DEPxR1
	&& InTriggerFF)
	{
	if(T && InTriggerFF)
	InTriggerFF ← 0; //if qualified event, clear FF
	
	Hold(DEPxR0); //Retain state of DEPxR0 (SEA)
	if(P==1)
	OutTrigger ← 1; //output 1 cycle pulse
	if(S==1)
	EPINT ← 1;
	
	}

FIG. 10I

1012 ↓ 1090 ↓

CONTROL VALUE	OPERATION

SPT10010
 	
LEA & Mask match.
 	
'T' bit used to specify special actions.
 	
When T==1, count reg treated as a semaphore which can respond to a DMA write address match. A DMA Write with an address that matches LEA & Mask causes inc of count, while a processor (SP/PE) match event causes dec of count amd an EXTOUT pulse.
If count==0 when match occurs, no decrement of count occurs and no EXTOUT pulse (basically no action).
	if(P==0)
	OutTrigger ← InTrigger; //Default is pass-through
	else
	OutTrigger ← 0; //Default is "no trigger" to output
	
	InTriggerFF ← 1; //Always enable event point for this code
	
	//Check for DMA access to increment count
	if(T==1)
	if((DMA Write Addr & DEPxR1)==(LEA & DEPxR1)//compare after DMA mux
	DEPxR2.H0 = DEPxR2.H0+1; //increment count
	
	//Check for SP/PE access
	if((DEPxR0 & DEPxR1)==(LEA & DEPxR1) && InTriggerFF)
	{
	if(DEPxR2.H0==0) //If match with count=0, carry on
	{ //(acts as if disabled)
	//do nothing
	}
	else
	{
	if(S==1 && P==1)
	OutTrigger ← 1; //Signal on OutTrigger
	
	if(T==0) //if not using DMA signaling...
	{
	if(DEPxR2.H0 == 1) // check for count reload
	{
	DEPxR2.H0 ← DEPxR2.H1; // reload count
	if(S==1 && P==0)
	EPINT ← 1; //if debug int selected, pulse signal
	}
	else
	DEPxR2.H0 = DEPxR2.H0 -1; //decrement count
	}
	else //else Using DMA signaling...
	{
	EXTOUT ← 1; Pulse external output 1 cycle
	DEPxR2.H0 = DEPxR2.H0 -1; //decrement count
	}
	
	}
	}

FIG. 10J

CONTROL VALUE	OPERATION

SPT10011
 	
SEA & Mask match.
 	
'T' bit used to specify special actions.
 	
When T==1, count reg treated as a semaphore which can respond to a DMA write address match. A DMA Read with an address that matches SEA & Mask causes inc of count, while a processor (SP/PE) match event causes dec of count and an EXTOUT pulse. If count==0 when match occurs, no decrement of count occurs and no EXTOUT pulse (basically no action).
	if(P==0)
	OutTrigger ← InTrigger; //Default is pass-through
	else
	OutTrigger ← 0; //Default is "no trigger" to output
	
	InTriggerFF ← 1; //Always enable event point for this code
	
	//Check for DMA access to increment count
	if(T==1)
	if((DMA Read Addr & DEPxR1)==(SEA & DEPxR1)//compare after DMA mux
	DEPxR2.H0 = DEPxR2.H0+1; //increment count
	
	//Check for SP/PE access
	if((DEPxR0 & DEPxR1)==(LEA & DEPxR1) && InTriggerFF)
	{
	if(DEPxR2.H0==0) //If match with count=0, carry on
	{ //(acts as if disabled)
	//do nothing
	}
	else
	{
	if(S==1 && P==1)
	OutTrigger ← 1; //Signal on OutTrigger
	
	if(T==0) //if not using DMA signaling...
	{
	if(DEPxR2.H0 == 1) // check for count reload
	{
	DEPxR2.H0 ← DEPxR2.H1; // reload count
	if(S==1 && P==0)
	EPINT ← 1; //if debug int selected, pulse signal
	}
	else
	DEPxR2.H0 = DEPxR2.H0 -1; //decrement count
	}
	else //else Using DMA signaling...
	{
	EXTOUT ← 1; Pulse external output 1 cycle
	DEPxR2.H0 = DEPxR2.H0 -1; //decrement count
	}
	
	}
	}

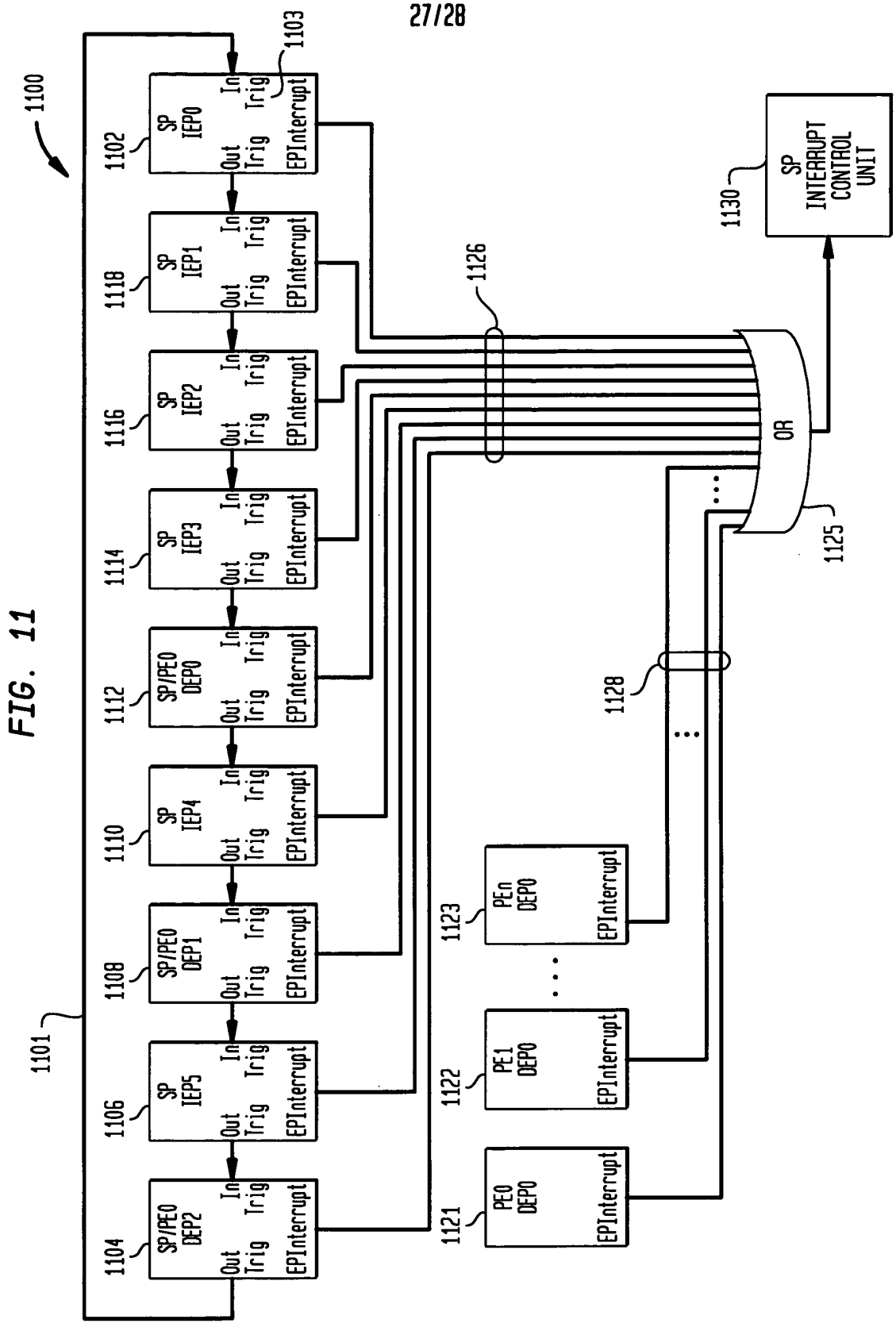
1100
1101
1102
SP IEP0
1103
1104
SP/PE0 DEP2
1106
SP IEP5
1108
SP/PE0 DEP1
1110
SP IEP4
1112
SP/PE0 DEP0
1114
SP IEP3
1116
SP IEP2
1118
SP IEP1
In Trig
Out Trig
EPInterrupt
1121
PE0 DEP0
1122
PE1 DEP0
1123
PEn DEP0
1125
OR
1126
1128
1130
SP INTERRUPT CONTROL UNIT

FIG. 11

FIG. 12A

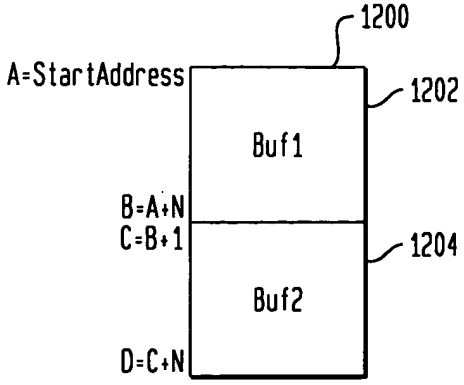

FIG. 12B

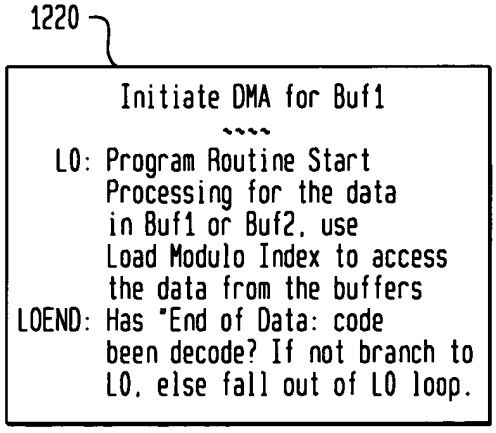

FIG. 12C